Modelización matemática en control de plagas

Seminario 1: el entorno de trabajo R

Contents

Para una mejor visualización de este documento se puede visitar https://jabiologo.github.io/web/tutorials/estadisticaBasica.html

Introducción al entorno R

¿Por qué R? R es un entorno y lenguaje de programación con un enfoque al análisis estadístico.

- R es una herramienta libre y gratuita para todos los sistemas operativos.
- R no es solamente un software para análisis estadístico.
- Es muy probable que tarde o temprano tengáis que utilizar esta herramienta.

Descarga e instalación de R y RStudio

- **R** lo podemos descargar en https://cran.r-project.org/ para los sistemas operativos Windows, macOS y Linux.
- **RStudio** lo podemos descargar en https://posit.co/download/rstudio-desktop/, también para cualquer sistema operativo

Una vez descargados, RStudio se vinculará con R automáticamente.

Un paseo por R y RStudio Es importante diferenciar entre R y RStudio. El software o lenguaje que utilizamos para los análisis estadisticos es **R**, por tanto podemos utilizar la consola de R por sí sola. **RStudio es un IDE** (del inglés *integrated development environment*), un entorno de desarrollo integrado, una "carcasa" que envuelve a R y facilita su uso. Es imporante familiarizarse con RStudio, ya que utilizaremos R a través de él. Tómate unos minutos para explorar sus diferentes paneles y personalizar su diseño:

- Diferencia entre *consola* y *script*
- Visualiza tu *entorno de trabajo*, tu ventana de *gráficos (plots)*, tus *librerías (paquetes)* cargadas, etc.
- También puedes cambiar el color de fondo, aumentar el tamaño de letra, etc.

Las `funciones()` Las **`funciones()`** son la "maquinaria" de R, las que realizan el trabajo. Se pueden identificar porque generalmente van seguidas de unos paréntesis entre los cuales se colocan sus **argumentos**. Los argumentos de una función son elementos que necesita esa función para ejecutarse y suelen ir separados por *comas* ,. Por ejemplo, existe una función que se llama "concatenar" (que en el lenguaje R se escribe `c()`), que simplemente sirve para unir en el mismo vector una serie de elementos (letras, números, etc.). Vamos a utilizar esa función para unir en un único vector una serie de números.

```
# La almohadilla se utiliza para incluir un comentario. Todo lo que se encuentre
# después de una almohadilla no se ejecutará en la consola.

# Unir los números 3, 7, 12 y 4 en un único vector
c(3, 7, 12, 4)
```

```
## [1]  3  7 12  4
```

- ¿Cuál es la función? ¿Cuáles son sus argumentos?

Imaginemos que ahora queremos hallar la media de esos números. Podemos utilizar otra función denominada `mean()` a la cual le introduciremos los numeros que hemos concatenado anteriormente como único argumento:

```
# Obtener la media de los números 3, 7, 12 y 4
mean(c(3, 7, 12, 4))
```

```
## [1] 6.5
```

Todas las funciones de R se alamcenan en "bibliotecas" o librerías (habitualmente denominados paquetes). Estos paquetes podemos entenderlos como cajas de herramientas donde se almacenan las herramientas que queremos utilizar. Hay algunos paquetes que vienen instalados y cargados por defecto en R. Sin embargo, otros los tenemos que descargar e instalar por primera vez y luego cargarlos en nuestra sesión cada vez que queramos utilizarlos. Por ejemplo, el paquete `lme4` se utiliza frecuentemente para ajustar modelos generales lineales mixtos. Podemos descargarlo, instalarlo y cargarlo en nuestra sesión de R de la siguiente manera:

```
# Descargamos e instalamos el paquete
install.packages("lme4")
# Cargamos el paquete en nuestra sesión
library(lme4)
```

Una de las grandes ventajas (o inconvenientes?) de R es que es un software libre, por lo que cualquiera puede desarrollar sus propios paquetes con las herramientas (funciones) que necesite y ponerlo a disposición de la comunidad de usuarios. Si tenéis curiosidad, aquí podéis encontrar un pequeño tutorial sobre como hacerlo.

Los objetos Los **objetos** en R son los contenedores donde almacenamos los resultados (outputs) de las funciones. Podemos identificarlos porque suelen aparecer por primera vez precediendo a los caracteres `<-`, que simbolizan una flecha que señala hacia la izquierda. Cada vez que se quiera crear un objeto se le ha de dar un nombre, el que queramos, aunque suele ser conveniente darle un nombre que tenga sentido. Por ejemplo, vamos a almacenar en un objeto que vamos a llamar "numeros" la concatenación de valores que creamos anteriormente:

```
# Almacenamos en un objeto llamado "numeros" el resultado de concatenar 3, 7, 12 y 4
numeros <- c(3, 7, 12, 4)

# Ahora podemos "llamar" a "numeros" para ver qué tiene dentro
numeros
```

```
## [1]  3  7 12  4
```

```
# También podemos utilizar a "numeros" dentro de otra función, por ejemplo mean()
mean(numeros)
```

```
## [1] 6.5
```

```
# Por último podemos guardar dentro de otro objeto el resultado de la linea anterior

m <- mean(numeros)
m
```

```
## [1] 6.5
```

Todos los objetos en R tienen una clase, que informa sobre el tipo de objeto que es. Por ejemplo, si es un vector de números será `numeric`, pero si lo que almacena son caracteres su clase será `character`. Hay muschísimas clases de objetos (¡incluso se pueden crear clases nuevas!). Conocer la clase de nuestros objetos es muy important, puesto que **algunas funciones necesitan que sus argumentos sean de una clase**

específica, y sino no funcionarán. Por ejemplo, no podemos hacer la media de las letras "a", "b" y "c", pero sí podremos hacer la media de los números 1, 2 y 3.

```
# Para averiguar la clase de un objeto usamos la función class()
class(numeros)
```

```
## [1] "numeric"
```

```
# Vamos a crear un objeto con los caracteres "a", "b" y "c"
letras <- c("a", "b", "c")

# Exploramos la clase de letras
class(letras)
```

```
## [1] "character"
```

```
# Vamos a intentar hacer la media de letras
mean(letras)
```

```
## Warning in mean.default(letras): argument is not numeric or logical: returning
## NA
```

```
## [1] NA
```

Entorno y directorio de trabajo Todos los objetos que vayamos creando o cargando en R se pueden visualizar en el panel `Environment` que suele situarse en la parte superior derecha en RStudio. En ese panel aparece información sobre los objetos, una previsualización o incluso, si el objeto lo permite, podemos hacer click en él y visualizarlos de forma intuitiva en formato "hoja de cálculo". También es importante ser conscientes del directorio de trabajo en el que estamos trabajando, esto es, la carpeta en la que se guardarán los ficheros que salgan de R, o la carpeta desde donde se cargarán los ficheros. Para saber nuestro directorio de trabajo actual utilizamos la función `getwd()`, mienstras que para cambiarla utilizaremos `setwd("mi_directorio_de_trabajo_nuevo")`.

```
# Exploramos mi directorio de trabajo actual
getwd()
```

```
## [1] "/home/javifl/github/web"
```

```
# Cambiamos el directorio de trabajo
setwd("/home/javifl/github/web/tutorials/estadisticaBasica_files")
```

Trabajo con `data.frame` Uno de los objetos que más vamos a utilizar en la práctica es el `data.frame`, que podría asimilarse a un fichero similar a lo que podríamos visualizar en una hoja de cálculo tipo Excel. Vamos a utilizar la función `read.csv` leer un archivo de texto plano delimitado por comas (CSV del inglés *"comma separated values"*) y ensayar algunas cosas con él. Podéis echar un vistazo a este archivo pinchando aquí.

```
# Guardamos el archivo en un obejto que denominaremos "datos"
datos <- read.csv("https://raw.githubusercontent.com/jabiologo/web/master/tutorials/estadisticaBasica_f

class(datos)
```

```
## [1] "data.frame"
```

```
# Podemos utilizar la función head() para explorar los primeros elementos de datos
head(datos)
```

```
##     peso    sex      hab      tri
## 1 419.35  Macho Agricola 47.87247
## 2 465.11 Hembra   Urbano 13.03228
## 3 434.17 Hembra   Urbano 18.05707
```

```
## 4 508.56  Macho Agricola 20.87707
## 5 514.75  Macho Agricola 15.29327
## 6 478.53  Macho   Urbano 15.52720
```

```
# Podemos utilizar la función str() para ver la estructura interna del data.frame
str(datos)
```

```
## 'data.frame':    300 obs. of  4 variables:
##  $ peso: num  419 465 434 509 515 ...
##  $ sex : chr  "Macho" "Hembra" "Hembra" "Macho" ...
##  $ hab : chr  "Agricola" "Urbano" "Urbano" "Agricola" ...
##  $ tri : num  47.9 13 18.1 20.9 15.3 ...
```

```
# O incluso podemos pedirle un resumen con summary()
summary(datos)
```

```
##       peso           sex                hab                tri
##  Min.   :330.0   Length:300         Length:300         Min.   :-4.733
##  1st Qu.:430.9   Class :character   Class :character   1st Qu.:18.955
##  Median :454.9   Mode  :character   Mode  :character   Median :25.853
##  Mean   :453.0                                         Mean   :25.780
##  3rd Qu.:475.9                                         3rd Qu.:32.421
##  Max.   :555.1                                         Max.   :52.502
```

Tenemos varias formas de manejar los datos contenidos en un `data.frame`. Por ejemplo, si queremos explorar alguna columna podemos utilizar el símbolo del dólar `$` para seleccionarla por su nombre. Por ejemplo `datos$peso`. Si lo que queremos es seleccionar un dato en concreto, podemos utilizar los corchetes `[]` para acceder a los elementos del `data.frame` indicando su posición en cuanto a filas y columnas separados por una coma con la forma `datos[fila,columna]`. Por ejemplo, si queremos seleccionar el dato que está en la fila 3 y columna 4, podemos emplear `datos[3,4]`. Si dejamos uno de los dos huecos en blanco, obtendremos toda la fila o toda la columna. Por ejemplo, si quremos todos los datos de la fila 7, podemos hacer `datos[7,]`. Veamos estos ejemplos.

```
# Visualizamos los nombres de las columnas de datos
colnames(datos)
```

```
## [1] "peso" "sex"  "hab"  "tri"
```

```
# Visualizamos los 15 primeros elementos de la columna peso
datos$peso[1:15]
```

```
##  [1] 419.35 465.11 434.17 508.56 514.75 478.53 429.03 463.79 445.34 412.27
## [11] 443.78 329.97 414.74 461.40 388.19
```

```
# Visualizamos el dato que está en la fila 3 y columna 4
datos[3,4]
```

```
## [1] 18.05707
```

```
# Visualizamos todos los datos de la fila 7
datos[7,]
```

```
##     peso    sex    hab      tri
## 7 429.03 Hembra Urbano 32.48139
```

Por último, vamos a ver una forma más moderna (y cómoda) de trabajar con los datos a través de `tydiverse`. Tydiverse es una coleccion de paquetes de R con una sintaxis en teoría más intuitiva que se centran en el manejo de datos. Utilizan el formato de "tuberías" o pipes mediante el símbolo `%>%`. Veamos un ejemplo.

```r
# Instalamos (en caso necesario) y cargamos las librerías de tidyverse
# install.packages(tidiverse)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
# Tomamos el objeto datos, y lo filtramos dejando tan solo los machos
datosMacho <- datos %>% filter(sex == "Macho")

# Visualizamos los primeros elementos del nuevo objeto
head(datosMacho)
```

```
##     peso   sex      hab      tri
## 1 419.35 Macho Agricola 47.87247
## 2 508.56 Macho Agricola 20.87707
## 3 514.75 Macho Agricola 15.29327
## 4 478.53 Macho   Urbano 15.52720
## 5 463.79 Macho Matorral 23.83045
## 6 412.27 Macho Agricola 46.89978
```

```r
# También podemos obtener una muestra aleatoria de un número de filas
muestra11 <- datos %>% sample_n(11)

# Número de filas de muestra11
nrow(muestra11)
```

```
## [1] 11
```

```r
muestra11
```

```
##      peso    sex      hab       tri
## 1  438.05  Macho   Urbano 31.157731
## 2  430.04  Macho Matorral 38.059647
## 3  478.66  Macho Agricola 24.890908
## 4  467.72  Macho Agricola 24.215332
## 5  500.88  Macho Matorral  1.600371
## 6  459.40  Macho Matorral 33.397504
## 7  426.85 Hembra   Urbano 29.326272
## 8  465.11 Hembra   Urbano 13.032283
## 9  445.34 Hembra Agricola 26.526576
## 10 461.32  Macho Agricola 24.847415
## 11 447.58  Macho Matorral 37.729169
```

La ayuda `help()` y preguntar a Google Otra de las ventajas de R con respecto a otros lenguajes de programación es que sus paquetes deben de cumplir una serie de estándares para poder estar en el repositorio "oficial" CRAN. Uno de los requerimientos es que las funciones de los diferentes paquetes deben de estar bien documentadas, es decir, debe existir un *manual* que indique cómo se usa cada una de las funciones del

paquete. Ese manual sigue siempre la misma estructura: nombre y descripción de la función, cómo se usa, los argumentos que necesita, el objeto que resulta al aplicar la función (Value), y unos ejemplos de cómo usar la función. Este manual o ayuda puede incluir más apartados, pero los mencionados suelen ser obligatorios. La forma de "llamar a la ayuda" en R es utilizando la función `help()`. Veamos un ejemplo:

```
# Vamos a intentar obtener la media del objeto letras
mean(letras)
```

```
## Warning in mean.default(letras): argument is not numeric or logical: returning
## NA
```

```
## [1] NA
```

```
# No entendemos por qué obtenemos este error... vamos a consultar la ayuda para
# entender qué necesita la función mean()
help(mean)

# Como vemos en el apartado Arguments, a la función mean() sólo se le pueden
# proporcionar objetos que sean de la clase numeric, logical, vectors, date,
# date-time y time_interval.
# Dado que la clase de letras es character, ahora entendemos por qué no podemos
# utilizar mean con letras.
```

Quizás la lección más importante es que normalmente ningún usuario habitual ha asistido a ningún curso especializado de R: la mayoría aprende a base de prueba/error y muchas horas delante del ordendaor con `ERROR` o `WARNINGS` en nuestra pantalla. Por eso es esencial no perder la paciencia y preguntar siempre a Google antes de preguntar al compañero que sabe (siemrpe hay alguien que ha tenido la misma duda antes que nosotros y ha dejado un comentario en un foro o blog). De esta forma interiorizaremos mucho mejor el funcionamiento de este lenguaje y aprenderemos mucho más rápido!